Searching Service Repositories by Combining Semantic and Ontological Matching

Tanveer Syeda-Mahmood, Gauri Shah
IBM Almaden Research Center
650 Harry Road, San Jose, CA
stf@almaden.ibm.com,gauris@almaden.ibm.com

Rama Akkiraju, Anca-Andrea Ivan, Richard Goodwin
IBM Watson Research Center
19 Skyline Drive, Hawthorne, NY
akkiraju@us.ibm.com,ivananca@gmail.com,rgoodwin@us.ibm.com

Abstract

Selecting the appropriate web service from a repository is a difficult problem as the terminology used to describe Web Services in repositories may not be identical to the one used in a query. In this paper, we explore the use of domain-independent and domain-specific ontologies to find matching service descriptions. The domain-independent relationships are derived using an English thesaurus after tokenization and part-of-speech tagging. The domain-specific ontological similarity is derived by inferencing the semantic annotations associated with web service descriptions. Matches due to the two cues are combined to determine an overall semantic similarity score. In addition, we integrate semantic and ontological matching with an indexing method, which we call attribute hashing, to enable fast retrieval of semantically related services. By combining multiple cues, we show that better relevancy results can be obtained for service matches from a large repository, than could be obtained using any one cue alone.

1. Introduction

Service oriented architectures, and web services in particular, offer the promise of easier system integration by providing standard protocols for data exchange using XML messages and a standard interface declaration language such as the web service description language (WSDL). This loosely coupled approach to integration provides encapsulation of service implementations, making it suitable for use with legacy systems and for promoting reuse by making external interfaces explicitly available in a WSDL description.

In this paper, we address the problem of identifying an appropriate web service for implementing a required function from a large collection of available web services. The problem arises in business transformation projects where new and modified business processes need to be implemented and there is a desire to reuse existing service implementations whenever possible. In the context of a large enterprise that was created through mergers and acquisitions, there are often thousands of applications and data repositories, each with associated services and interface definitions. Since these interfaces were created for different organizations, over different periods of time and for different initial purposes, it is unlikely that they use a common set of terms to name services and parameters.

In performing the search to identify existing services that can be used to implement a required service, we need to address two fundamental issues. First, we need a way to match the requirements and capabilities of services i.e, identify similarity between services. Secondly, we need an efficient way to search the service repositories so that we can avoid looking at obviously irrelevant services.

Identifying similarity between services is a difficult problem because the terminology used to describe Web Services may not be identical to the one used in a query. In addition, structure and type information present in service description will have to be taken into account. As a result, a full-text search of the repositories rarely retrieves relevant matches.

In this paper, we explore the use of domain-independent and domain-specific ontologies to find matching service descriptions. The domain-independent relationships are derived using an English thesaurus after tokenization and part-of-speech tagging. The domain-specific ontological similarity is derived by inferring the semantic annotations associated with web service descriptions. Using an ontology. Matches due to the two cues are combined to determine an overall similarity score. In addition, we integrate semantic and ontological matching with an indexing method, which we call attribute hashing, to enable fast retrieval of semantically related services. By combining multiple cues, we show that better relevancy results can be obtained for service matches from a large repository, than could be obtained using any one cue alone.

The rest of the paper is organized as follows. In Section 2, we motivate the need for modeling domain-specific and domain-independent semantics for identifying similarity in services. In Section 3, 4, and 5, we present matching algorithms for each of the domain-specific and domain-independent cues and their combination. In Section 6, we discuss the indexing of large service repositories to perform an efficient search using a new indexing technique called attribute hashing. Section 7 presents experimental results on a large service repository assembled from business process integration scenarios, and provides a comparison with full-text searching of services repositories. Section 8 discusses related work.

2 Use of semantics to define service similarity

As in document retrieval, searching for services should be on the notion of similarity rather than identical matches as we are unlikely to find exactly matching descriptions. However, straightforward information retrieval techniques that are based on frequency of occurrence of terms in a document cannot be used directly, since we are more likely to find attributes in a service definition present in many other services, rather than multiply occurring within a service description itself. Further, the structure and type information present in service descriptions need to be taken into account.

So how can similarity between services be discovered? This is a difficult problem, in general. Ideally, we would like to match two services if their capabilities and requirements are the same. The current industry standards such as WSDL describe a service as a collection of operational interfaces and their type specification together with deployment information. These specifications are limited in their ability to express the capabilities and requirements of the services themselves. Even in interface specifications, the terms used to describe matching services may be semantically related but not necessarily identical. In other cases, the terms used may not appear to be semantically related at all, unless domain-specific information is used to interpret their relationships.

Figure 1 shows an example of two related services for describing inventory. The interface descriptions (inputs and outputs of operations) are shown here in the form of a tree for purpose of illustration, and are extracted from their WSDL descriptions. The closely related terms shown by the red arrows could be discovered using an English thesaurus after appropriate tokenization, word expansion and part-of-speech tagging. These include obvious cases such as terms (OrganizationID,OrgID) as well as non-obvious ones such as (InventoryType, StockType).

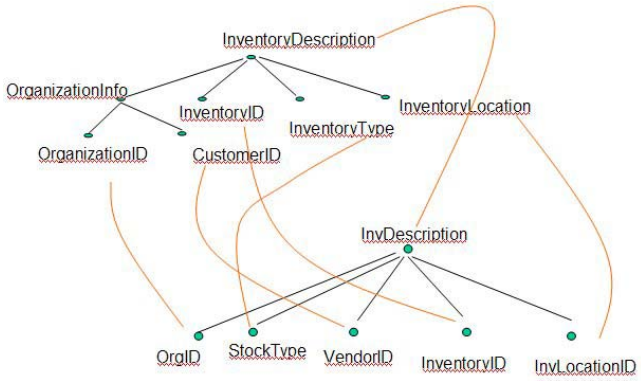

Figure 1. Illustration of related services based on semantically related terms used in their interface descriptions.

Figure 2 shows an example of two related services in which the relationships between terms could not be discovered using an English thesaurus alone. Figure 2a shows an extract of a WSDL description of an inventory checking service of an electronics company XYZ. It takes the requested item's part number, delivery date and the requested quantity as inputs and returns the quantity available to be delivered on the requested date. The operation tag lists the inputs and outputs the service takes. The actual inputs and outputs are expanded in QueryAvailabilityServiceRequest and QueryAvailabilityServiceResponse message tags. Figure 2b, on the other hand, shows another web service offered by ABC Inc. checking inventory. However, ABC calls it CheckInventoryService and its inputs and outputs are different from the ones offered by XYZ company's QueryAvailabilityService. ABC's service requires a Universal Product Code instead of a manufacture's part number. The term dueDate is used rather than DeliveryDate and NumberOfItems is used rather than Quantity. Also, ABC's service just returns an ItemAvailabilityConfirmation, which is true if the requested quantity is available and false otherwise. On the other hand, XYZ's service indicates when a request can be partially filled, by returning the number of available items. As can be seen, there are differences in the interfaces of the services. However, if the objective is to find a service that gives information about the availability of a given part, semantically, both services should be matched. This type of match cannot be determined using an English language thesaurus alone. Domain-specific information is needed to interpret the relationships. For example, in this domain due date and delivery date are synonyms, UPC expands to Universal Product Code, a UPC and a manufacture's part number are both unique identifiers for a product, and UPCs is a subset of EAN Codes. A domain specific ontology, such as the one shown in figure 3, is required to record such relationships.

```
.
<message name="QueryAvailabilityServiceRequest">
      <part name="partNumber_in" type="xsd:string" />
              <part name="deliveryDate_in" type="xsd:string" />
      <part name="quantityRequested_in" type="xsd:string"/>
  </message>
  <message name="QueryAvailabilityServiceResponse">
     <part name="quantityAvailable_out" type="xsd:string" />
  </message>

  <portType name="QueryAvailabilityService">
     <operation name="queryAvailabilityService" >
        <input message="tns:queryAvailabilityServiceRequest"
name="queryAvailabilityServiceRequest"/>
        <output message="tns:queryAvailabilityServiceResponse"
name="queryAvailabilityServiceResponse"/>

     </operation>
   </portType>
.....
```

(a)

```
.....

<message name="CheckInventoryService ">
      <part name="UPC_in" type="xsd:string"/>
            <part name="duedate_in" type="xsd:string"/>
      <part name="numberOfItems_in" type="xsd:string"/>
  </message>
  <message name="CheckInventoryServiceResponse">
      <part name="itemAvailabilityConfirmation_out"
type="xsd:string"/>
  </message>
  <portType name="CheckInventoryService">
     <operation name="checkINventoryService" >
        <input message="tns:checkInventoryServiceRequest"
name="checkInventoryServiceRequest"/>
        <output message="tns:checkInventoryServiceResponse"
name="checkInventoryServiceResponse"/>
     </operation>
   </portType>
.....
```

(b)

Figure 2. Illustration of the need for modeling of domain-specific information for finding relationships between services.

The above example shows that a domain independent source of clues gives us a breadth of coverage for common terms, while a domain specific ontology gives a depth of coverage by providing clues based on industry and application specific terms and relationships. By combining these sources, we hope to improve both precision and recall. This is a key idea in the work reported in this paper.

3. Finding related services using domain-independent semantics

In this section, we discuss the use of domain-independent semantics to find related services. Finding semantic relationship between attributes is difficult, in general, because (1) attributes could be multi-word terms (e.g. Customer2Identification, PhoneCountry, etc. which require tokenization. Any tokenization must capture naming conventions used by programmers to form attribute names. (2) Finding meaningful matches would need to account for senses of the word as well as their part-of-speech through a thesaurus. (3) Multiple matches of attributes must be taken into account, and (4) finally, the structure/type information must be exploited so that operations match to operations, messages to messages, etc.

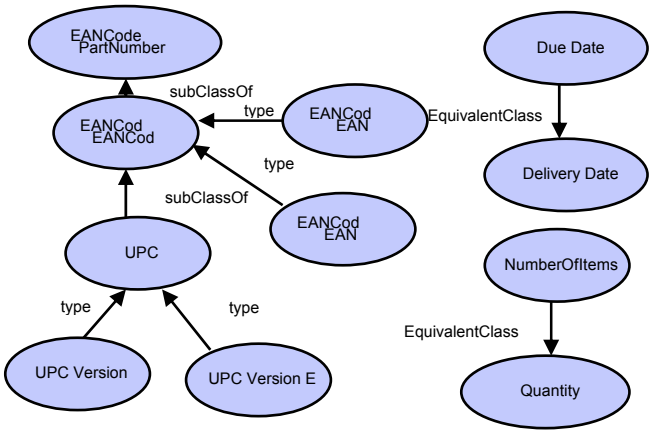

Figure3: A simple Electronics domain ontology

3.1 Capturing domain-independent similarity of service attributes

We capture name semantics using a technique similar to the one in [9][14]. Specifically, multi-term query attributes are parsed into tokens. Part-of-speech tagging and stop-word filtering is performed. Abbreviation expansion is done for the retained words if necessary, and then a thesaurus is used to find the similarity of the tokens based on synonyms. The resulting synonyms are assembled back to determine matches to candidate multi-term word attributes of the repository services after taking into account the tags associated with the attributes. The details are described below:

Work Tokenization

To tokenize words, we exploit common naming_conventions used by programmer analysts. In particular, we find_word boundaries in a multi-term word attribute using changes in font, presence_of delimiters such as underscore, spaces, and numeric to alphanumeric_transitions. Thus words such as CustomerPurchase will be separated into_Customer and Purchase. Address_1, Address_2 would be separated into Address,1 and Address,2 respectively. This allows for semantic matches of the attributes.

Part-of-speech tagging and filtering

Simple grammar rules are used to detect noun phrases and adjectives. Stop-word filtering is performed using a pre-supplied list. We have used common stop words in the English language similar to those used in search engines, including words such as and, or, the, etc.

Abbreviation expansion

The abbreviation expansion uses domain-independent as well as domain-specific vocabularies. It is possible to have multiple expansions for a candidate words. All such words and their synonyms are retained for later processing. Thus, a word such as CustPurch will be expanded into CustomerPurchase, CustomaryPurchase, etc.

Association of tag type with attributes

All resulting tokens and their words are associated back with their tags in the service descriptions. Thus tokens from operation names would be tagged with an operation type and tokens from messages with a message type and so on.

Synonym search

We used the WordNet thesaurus [16] to find matching synonyms to words. Each synonym was assigned a similarity score based on the sense index, and the order of the synonym in the matches returned.

Match generation

Consider a pair of candidate matching attributes (A, B) from the query and repository services respectively. These matching attributes could be a pair of inputs to be matched from a service request and an available service from a repository. Let A, B have m and n valid tokens respectively, and let S_{y_i} and S_{y_j} be their expanded synonym lists based on ontological processing. We consider each token i in source attribute A to match a token j in destination attribute B where $i \varepsilon S_{yi}$ and $j \varepsilon S_{yi}$. The semantic similarity between attributes A and B is then given by

$$Sem(A, B) = 2*Match(A,B)/(m + n) \quad (1)$$

where Match(A, B) are the matching tokens based on the definition above. The semantic similarity allows us to match attributes such as (state, province), (CustomerIdentification, ClientID), (CustomerClass, ClientCategory), etc.

3.2 Ranking services based on domain-independent similarity

Our approach to ranking services using domain-independent similarity is based on the rationale that related services in the repository have an overwhelming number of attributes that are semantically related to query attributes, so that indexing based on query attributes could point to relevant matching services.

Give a query WSDL file and a set of WSDL files in a repository, it is reasonable to assume that the best matching services are those that have a large number of semantically related attributes. Let there be k services in the repository. Let h1, h2,…, hk be the number of attributes of services S1, S2,…, Sk that can be matched with query attributes using the semantic match score of Equation 1. Let n1, n2,…, nk be the number of attributes present in services S1, S2,...,Sk. Then the overall match of the query service to a repository service is given by

$$M_{sem,i} = min\{h_i/n_i, h_i/|Q|\} \quad (2)$$

The best matching service to a query service is then given by

$$S_{sem} = max\{M_{sem,i}\} \text{ for all } 1 \le i \le k \quad (3)$$

Further, the values of M_i can be sorted to get a ranked list of matching services. By taking $max\{M_{sem,i}\}$, we look for those matches that have the lowest number of unmatched attributes

relative to their service size. Thus if we had a query service of 5 attributes with matches to all 5 query attributes in two services S1 and S2 with the number of attributes 10 and 6 respectively, then S2 is preferred over S1 because of the larger ratio of number of attributes matched.

4. Finding related services using domain-specific semantics

Following the argument in Section 2, while domain-independent cues give us a breadth of coverage for common terms, we still need domain-specific ontological information to find deeper relationships based on industry and application specific terms that could not otherwise have been provided through language-dependent thesauruses. In this section, we discuss the use of domain-specific ontologies to find similar services.

4.1 Domain-specific ontologies

We create domain-specific ontologies using OWL[5]. Specifically we use a semantic network-based ontology management system known as SNoBASE (Lee et al 2003) that offers DQL-based (DQL 2003) Java API for querying ontologies represented in OWL. The OWL-specified ontologies loaded into SNOBASE are parsed to populate its internal data store with facts and instances. A simple domain-specific ontology that models the relationships between electronic parts is indicated in Figure 3. As can be seen, four different types of relationships, namely, subClassOf(A,B), subClassOf(B,A) – which is essentially superClassOf, type (A,B) – which is instanceOf, and equivalenceClass(A,B) are modeled where A and B are two given concepts. Larger ontologies may model more relationships.

4.2 Finding related terms in an ontology

Given a domain-specific ontology and a query term, the related terms in an ontology are found using rule-based inference. In the SNoBASE system we used, IBM's ABLE (Bigus et al 2001) engine for inference. The ABLE library includes rule-based inference using Boolean and fuzzy logic, forward chaining, backward chaining etc. Rule sets created using the ABLE Rule Language can be used by any of the provided inference engines, which range from simple if-then scripting to light-weight inferencing to heavy-weight AI algorithms using pattern matching and unification [1].

4.3 Annotating domain-specific information in services

In order to use domain-specific ontological information, references to the ontology must be present in the service description. The standard WSDL specification does not have a mechanism to denote such ontological information and hence must be augmented before such information can be used to determine matching services. The subject of semantic annotation is an active area of research in the semantic web community with languages such as OWL-S , WSDL-S, etc. In this paper, we have adopted the WSDL-S specification due to the following reasons. First, users can describe, in an upwardly compatible way, both the semantics and operation level details in WSDL- a language that the developer community is familiar with. Secondly, by externalizing the semantic domain models, we take an agnostic approach to ontology representation languages. This allows Web service developers to annotate Web services with their choice of ontology language (such as UML or OWL) unlike in OWL-S. This is significant because the ability to reuse existing domain models expressed in modeling languages like UML can greatly alleviate the need to separately model semantics. Finally, it is relatively easy to update the existing tooling around WSDL specification to accommodate our incremental approach.

```
……..
xmlns:wssem="
http://www.myschema.com/schemas/2004/wssem"
xmlns:ElectronicsOntology="http://www.standards.com/ontologie
s/ElectronicsDomain.owl" >

 <message name="CheckInventoryServiceRequest">
       <part name="iUPC_in" type="xsd:string"
wssem:modelReference="ElectronicsOntology#UPC"/>
                 <part name="dueDate_in" type="xsd:string"
wssem:modelReference="ElectronicsOntology#DeliveryDate"/>
       <part name="numberOfItems_in" type="xsd:string"
wssem:modelReference="ElectronicsOntology#Quantity"/>
  </message>

  <message name="CheckInventoryServiceResponse">
       <part name="itemAvailabilityConfirmation_out"
type="xsd:string"/>
  </message>

……..
```

Figure 4: An excerpt of a semantically annotated WSDL for inventory checking service in figure 2b.

Using the WSDL-S specification, we annotate elements in the WSDL file using the attribute *wssem:modelReferenc*es. Its value is an OWL ontology concept specified by the name of the ontology and the relevant ontological term. Such an annotated WSDL file corresponding to the web service description of Figure 1 is shown in Figure 4. In this figure, the input UPC_in which is of type xsd:string has an annotation that points to the concept Universal Product Code in the Electronics Ontology.

4.2 Capturing domain-dependent similarity of service attributes

Given an ontology-annotated attribute in a query service, the potential matching attributes from services in the repository can be found as follows. The query attribute is used as a fact for inference by the SNOBASE system against the corresponding ontology. The matches returned are a set of related concepts along with distance scores representing distance between concepts. Each related concept models a specific type of relationship. In our case, we model EquivalentClass, subClassOf, superClassOf, and type as shown in Figure 3. We use the simple scoring scheme shown in Table 1 to compute distance between related concepts in the ontology. The discretization of the score into three values (0, 0.5, 1.0) gives a coarse idea of semantic separation between ontological concepts. For example, in the electronics domain ontology shown in Figure 2, concepts DueDate and DeliveryDate have a distance of 0 while EANCode and UPC have a distance of 0.5. More refined scoring schemes are possible, but the current choice works well in practice without causing a deep semantic bias. Thus given a query attribute with an ontological annotation term 'DueDate' will return ontologically matching concepts as the terms (DeliveryDate) while a query term "UPC" will return as related concepts (EAC code, Part Number, EAN8,

EAN13,UPCversion A, UPC version E) using inference in the ontology of Figure 3. If these terms were the associated annotations with service attributes, then the corresponding attributes can also be identified through the related concepts retrieved. In practice, we can choose a suitable threshold T so that all related concepts with distance scores above T can be ignored. From now on, we use the term related concepts to refer to such filtered attributes with scores above the threshold T.

Concept Pair	Relationship	Distance Score
(A,B)	EquivalentClass	0
(A,B)	RDFType	0
(B,A)	SubClassOf	0.5
(A,B)	SubClassOf	0.5
(A,B)	Other	1

Table 1: Distance scoring scheme in our ontology matcher

4.4 Ranking services based on ontological matching

As with domain-independent semantics-based ranking, the ontology-based matcher takes as input a query WSDL file and a set of WSDL files. The output is a subset of the original list of WSDL files, ordered by their relevance. Note that unlike in the case of domain-independent semantic matching, not all attributes in a service description may be annotated with ontological terms. Thus a separate normalization is necessary.

Let there be k services in the repository. Let h1, h2,…, hk be the number of annotated attributes of services S1, S2,…, Sk whose ontological concepts are related to those of corresponding query attributes (i.e whose ontological match score exceeds the threshold T). Let n1, n2,…, nk be the number of ontologically-annotated attributes present in services S1, S2,...,Sk. Then the overall match of the query service to a repository service S_i is given by

$$M_{ont,i} = min\left\{ \frac{\sum_j c_{ij}}{n_i}, \frac{\sum_j c_{ij}}{|Q|} \right\} \quad \text{..... (4)}$$

Where $c_{ij} = 1 - dist(i,j)$ and dist(i,j) is the ontological distance between the jth term in service S_i and a corresponding query term. The best matching service to a query service is then given by

$S_{ont} = max\{M_{ont,i}\}$ for all $1 \le i \le k$..... (5)

Further, the values of M_i can be sorted to get a ranked list of matching services. By taking $max\{M_i\}$, we look for those matches that have the lowest number of unmatched attributes relative to their service size as before.

5. Combining domain-specific and domain-independent ranking of services

The combination of domain-specific and domain-independent cues to find related services can be done at either an inner-level during the attribute similarity computations or after the individual rankings are produced. In either case, a winner-take-all scheme makes sense in which the best possible score (ontology-wise or semantic-matching-wise) is percolated up for the ranking computations. This scheme for the case of ranking already retrieved matches is described below.

For each potential matching service Si, let Msem,i be the matching score using semantic matching. Let Mont,i be the matching score using ontological matching. Then a new score can be computed for each potential matching service Si as

$$M_i = \max\{M_{sem,i}, M_{ont,i}\} \quad \text{.....(6)}$$

The best matching service can again be given by

$$S_{best} = \max\{M_i\} \quad \text{.....(7)}$$

6. Attribute hashing

In the above formulation of the ranking of services, all services attributes would have to be searched for each query service to find potential matches and to assemble the overall match results. We now present an efficient indexing scheme, called attribute hashing, that avoids detailed search of the service repositories while ensuring at the same time that no extra false positives or negatives are made for the same choice of ranking thresholds.

To understand the role of indexing, let us consider a service repository of 500 services. If each service has about 50 attributes (quite common for enterprise-level services), and 2 to 3 tokens per word attribute, and about 30 synonyms per token, the semantic matching alone would make the search for a query of 50 attributes easily around 50 million operations per query!

Indexing of the repository schemas is, therefore, crucial to reducing the complexity of search. Specifically, if the candidate attributes of the repository schemas can be directly identified for each query attribute without linearly searching through all attributes, then significant savings can be achieved.

6.1 Attribute hashing

We now present attribute hashing, an efficient indexing scheme that achieves the desired savings in search time. The key idea in attribute hashing can be explained as follows. Let ‘a’ be an entity derived from a repository service description. Let F(a) be the set of related entities of ‘a’ in the entire service repository (also called feature set here). In the case of domain-independent semantics ‘a’ refers to a token and F(a) is the set of synonyms of ‘a’. In the case on ontological matching, ‘a’ refers to an ontological annotation term, and F(a) are the ontologically related concepts to a (eg. terms related by subclass, equivalenceClass, is-a, etc. relationships) . Now, given a query entity q derived from a query service Q, q is related to a iff q ε F(a). Thus instead of indexing the set F(a) using the attribute a as a key as may be done in normal indexing, we use the terms in the set F(a) as keys to index a hash table and record ‘a’ as an entry in the hash table repeatedly for each such key. The advantage of this operation is that since q ε F(a), q is indeed one of the keys of the hash function. If this operation is repeated for all entities in the service repository, then each hash table entry indexed by a key records all entities whose related term set includes the key. *Thus indexing the hash table using the query entity q directly identifies all related entities from the service repository without further search*! Once the related entities per query term can be identified this way, a histogram of hits for each service contributing to the match can be taken and ranked list of

matching services can be obtained after suitable normalization. This is the key idea of attribute hashing.

Of course, this is done at the cost of redundant storage (the entity 'a' is stored repeatedly as an entry under each relevant key). However, with the growth of computer memory, storage is a relatively inexpensive tradeoff.

The purpose of attribute hashing is to identify relevant services. Exact ranking of services can be then be performed using detailed matching computations with domain-specific and domain-independent semantics and their combination as described in Sections 3, 4 and 5. Not that since a single query term can match multiple terms of the same service in the repository, this can lead to multiple counting of the same service hit. To avoid this, we take a greedy approach, i.e. the first time a query term finds a match in a service, further matches per service are not counted. We have shown earlier that such a choice still provides a lower bound on the size of the actual match between a query and repository service. This is still sufficient for retaining good service matches if we choose match thresholds based on the lower bound since the actual number of matching attributes may, in fact, be higher. Also, for purposes of pruning the set of irrelevant services, it is sufficient to know the number of matching attributes rather than their extent of similarity as given by distance scores. Thus distance scores will initially be ignored in attribute hashing, and will be reutilized later during detailed matching on the pruned set of services.

6.2 Offline index creation

More formally, let f_i be the feature set computed from an entity a_i. Let O_j represent all relevant indexing information corresponding to the entity a_i which in our case is given by

$$O_j =< t_j, c_{mj}, w_k, b_l, S_m > \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(8)$$

where t_j is the entity whose feature is f_i . In case of domain-independent matching, t_j is the token whose synonym is f_i , while in the case of domain-dependent matching t_j is a ontological term that is related to the term f_i using one of the relations defined in the ontology. Further c_{mi} is extent of similarity measured as 1- match distance. The domain-independent matcher uses a simple model of similarity for synonyms so that c_{mi} =1 for all synonyms of a token. Future work will be similarity scores of synonyms using the SureWord thesaurus. For domain-dependent matching c_{mi} is the cost of matching the ith ontological term t_j in service S_m to the feature term f_i.

In Equation (8) the term w_k refers to the actual service attribute from which the term t_j is derived. In the case of domain-dependent matching, this is the service attribute which was annotated with the ontological term t_j, while in the case of domain-independent matching, it is the word from which the token t_j is derived as explained in Section 3.1. Next b_l refers to the type tag associated with the term to allow for constrained matching based on tag information (operations to operations, messages to messages, etc.) Finally, the service information S_m is retained so as to allow the selection of appropriate services.

Since f_i can be the related term for many terms derived from attributes in the service repository, we could collect all such entries in a hash table and record all associated indexing information to be indexed by the feature key as:

$$H(f_i) = \{O_1, O_2, \ldots.\} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(9)$$

The hash table is constructed in an off-line index creation step in which all repository services are processed to extract multi-term word attributes and their associated ontological terms, if any. Each multi-term word attributes is tokenized, expanded and part-of-speech tagged as described in Section 3.1 and its synonyms are derived. Similarly, each ontological term is used to look up the corresponding domain-specific ontology to retrieve related terms. All such synonyms and related terms form the feature set F={ f_i }. A hash table is formed by recording all the indexing information described in Equation 8 to be indexed using the keys in F.

6.3 Ranking services using attribute hashing

To find related services using attribute hashing, query services are processed in a manner similar to the repository services during the offline index creation stage. While the basic unit of indexing is a term (token of a word attribute or ontological term), the histogram of hits is recorded per attribute and per service in the repository. Each query term q_i in query service Q with associated indexing information, $Oqi =< w_{qk}, b_{ql} >$ is used to index the hash table and the service hits are recorded. Following the argument in Section 6.1, we use the greedy approach to pick the first set of matching terms to query terms and count only the number of matches and not their distance scores for similarity. To avoid double counting, we record the hits to attributes using two arrays D and D' where D (w_{qk}) =0 if the query attribute w_{qk} has not been matched to any service attribute in the repository based on domain-independent cues. Similarly D'(w_{qk}) = 0 if the query attribute has not been matched to any service attributes based on domain-dependent cues. The entries D(w_k) and D'(w_k) for any service attribute can be defined similarly. The detailed algorithm for attribute hashing is given in Figure 5.

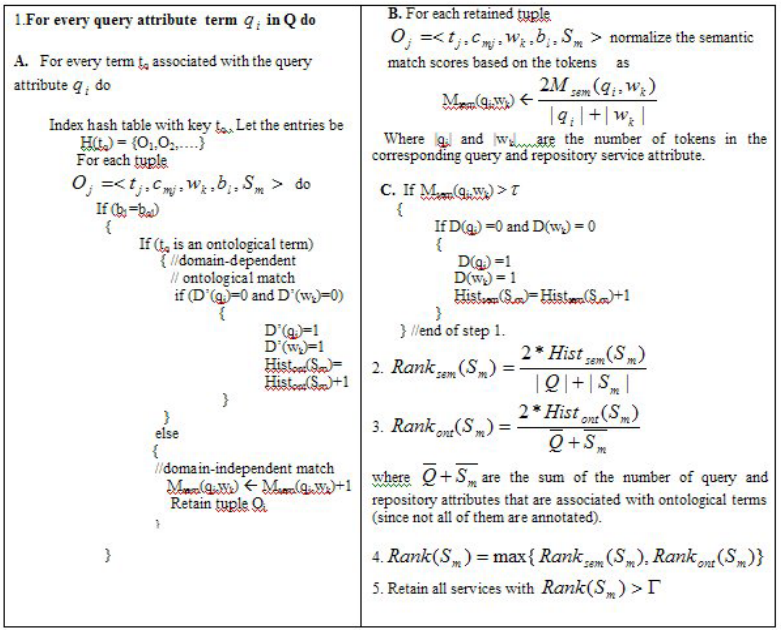

Figure 5. Algorithm for ranking services using attribute hashing.

7. Results

To demonstrate retrieval of web services based on domain-independent and domain-dependent semantics, we tested on a large

repository of industrial-strength web services drawn from about 500 business objects designed for Oracle, PeopleSoft and SAP applications as well as 473 WSDL documents assembled from legacy applications such as COBOL copybooks and from the general services offered on xmlmethods.com. Each of the services has embedded schemas that were rather large containing 100 or more attributes.

We measured the performance of our ranking scheme along three dimensions, namely, (a) precision versus recall for various choice of ranking thresholds Γ, (b) reduction in search time due to indexing using attribute hashing, and (c) role of domain-independent and domain-dependent semantics for retrieving relevant services. We also compared its performance to two other techniques of searching schemas, namely, full-text indexed search, and lexical match search. Full-text search engine for these repositories was developed by creating an inverted index of all the words extracted from services and computing the histogram of service hits using every query word to index the full-text index.
The second method we implemented was to illustrate the power of semantic search techniques over lexical match methods. In this we keep the indexing and search the same as our approach, but replace the domain-independent and domain-dependent matching computation described in Section 3 and 4 with a lexical similarity measure given by the edit distance of two strings representing the word attributes (no tokenization was done).

7.1 Precision-Recall Studies
To conduct the precision and recall we manually recorded the number of matching services in the repository for a sample of 40 queries. Figure 6 show the average precision versus recall using the three different methods of ranking services tried, namely, full-text indexing, lexical matching, and domain-independent semantic matching. The average precision and recall was recorded for various choice of the ranking threshold Γ between 0 and 1.0. From these figure, we can observe that domain-independent semantic matching has better recall than full-text search. In practice, we observed that the precision of semantic ranking is lower but the recall is higher. This can be improved when domain-dependent semantics are combined for ranking as we will show in Section 7.3. From this figure, we can also select an appropriate threshold for ranking. For example, by choosing a threshold of $\Gamma = 0.4$ we can obtain 80% recall and 60% precision using domain-independent semantic matching.

7.2 Indexing performance
Our approach to ranking has a large off-line one-time index creation overhead for the repository (periodically updated), but the search time is considerably smaller. Figure 7 shows the relative time performance of the ranking by attribute hashing in comparison to full-text index-based ranking and lexical match-based ranking. As can be seen, the search time is considerably smaller using attribute hashing.

We tested the indexing performance of the hashing scheme by noting the fraction of the repository touched during search. Using attribute hashing, the complexity of search reduces significantly, as only matching terms are explored. In fact, our experiments show that on average a 90-95% reduction in search is achieved by the use of attribute hashing. The entire repository consisting of over 100,000 total attributes indexes in less than two minutes on an Intel M-Pro 2 GHz Pentium with 1G RAM, and the matching services for queries are retrieved instantaneously.

7.3 Role of domain-dependent semantics
To assess the role of domain-dependent semantics, we ran the attribute hashing-based ranking using the same queries but with domain-independent thesaurus information alone in one case, and then with combination with ontological information as described in Figure 5. We used as input a set of 30 WSDL files annotated with OWL notions from the various ontologies.

Figure 8 illustrates the comparison results, as the ratio of combined matching score to domain-independent semantics-based score for matches returned averaged across the queries. As can be seen from this figure, the combined matcher improves the precision of matches by raising the rank of those services that match through the use of ontological information even though they may not have semantically similar terms. Thus the ranked order of matches returned can be different using the combined scheme. Finally, Figure 9 summarizes the relative effect of each cue in determining matches to sample queries. As can be seen from Figure 9, using ontology alone is not sufficient, while the performance of the semantic matcher is best when combined with ontology matching.

8.Related Work

The work presented in this paper is closely related to three areas, namely, UDDI registries, ontological matching, and document retrieval applications to web services.

8.1 UDDI registries
The notion of search through repositories has been popular in web services. Web service schemas are published to a public or private UDDI registry. The design of UDDI allows simple forms of searching and allows trading partners to publish data about themselves and their advertised Web services to voluntarily provide categorization data. Several companies are trying to put forward UDDI registries [8] including IBM and HP.

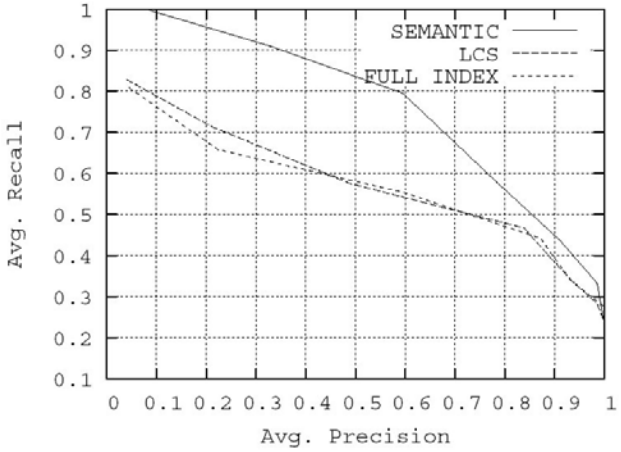

Figure 6. Illustration of precision and recall during search for matching services.

The three predominant ways of searching metadata repositories are (a) visual browsing through categories (b) keyword searches, and (3) XPath expressions. Visual navigation relies on a priori categorization of the services as in UDDIs, a laborious and

inexact process where a misclassification can lead to a false negative or a false positive. Keyword-based search techniques use information retrieval methods to do a full-text search of the underlying repository. Full-text search of web services based on a few keywords, however, can retrieve a number of false positives since the same keywords may occur in different services possibly within a different context and structure. Finally, XQuery specifies search through XPath expressions that capture the structure of the XML documents during navigation and search. While such structured queries can find exact matches, they are more difficult to use for similarity searches. Further, they require a priori knowledge of the schema to construct path queries.

8.2 Web service search

Recently, clustering and classification techniques from machine learning are being applied to the problem of web service matching and classification at either the whole web service level [10] or at the operation level [9]. In [10] for example, all terms from portTypes, operations and messages in a WSDL document are treated as a bag of words and multi-dimensional vectors created from these bag of words are used for web service classification. Although this type of classification retrieves matches with higher precision that full-text indexed search, the overall matches produced, however, do not guarantee a match of operations to operations, messages to messages, etc. The paper by Madhavan et al [9] addresses this aspect by focusing on matching of operations in web services. Specifically, it clusters parameters present in input and outputs of operations (i.e. messages) based on their co-occurrence into parameter concept clusters. This information is exploited at the parameter, the inputs and output, and operation levels to determine similarity of operations in web services. All these approaches use simplistic web services available on the web. Since there are only a handful of parameters per operation, and only partial matches of parameters can be expected in realistic web services. Thus it is not clear how this method scales to industrial strength web services that have few parameters each modeled though by complete XSD schemas wherein lies the true information for matching parameters.

8.3 Semantic matching

The problem of automatically finding semantic relationships between schemas has also been addressed by a number of database researchers lately [12][11]. The notion of elemental and structural level schema matching has been present in the METEOR-S project, where the engine can perform both element and structure level schema matching for Web services. The element level matching is based on a combination of Porter Stemmer for root word selection, WordNet dictionary for synonyms, abbreviation dictionary to handle acronyms and NGram algorithm for linguistic similarity of the names of the two concepts. The schema matching examines the structural similarity between two concepts. Both element match score and schema match score are then used to determine the final match score.

8.4 Ontological matching

One of the earliest ontology-based semantic matchmaking engines is Sycara et al MatchMaker [13] that is available on the Web as a service. In addition to utilizing a capability-based semantic match, the engine also uses various other IR- based filters thereby reducing the number of false positives. Another related effort is Racer [15], that focuses solely on a service capability-based semantic match for application in e-commerce systems. Method for semantically enhancing the service capabilities in UDDI was proposed earlier in [1]. In a related effort, Patil et al have developed MWSAF, a web service annotation framework [14]. In their work, they generate recommendations for automatically annotating WSDL documents. To accomplish this they match XML schema used by the WSDL files with ontologies by creating canonical schema graphs.

9. Conclusions.

In this paper, we explore the use of domain-independent and domain-specific ontologies to find matching service descriptions. The approach taken in this paper fundamentally differs from previous work in the following respects. First, it combines domain-dependent ontological information with domain-independent semantics for matching. Both name and type information is retained to select possible matches using either cue. Secondly, it presents an efficient indexing scheme to allow scalability of search across large service repositories. Finally. we perform extensive experiments to evaluate our approach on industrial strength web services used in enterprise application integration that often have large schemas (in excess of 100 attributes) and demonstrate the effectiveness of attribute hashing over straightforward document retrieval approaches in retrieving precise matches to web services with minimum search.

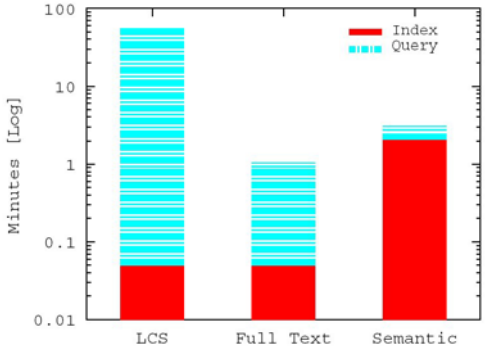

Figure 7: Illustration of search reduction due to attribute hashing.

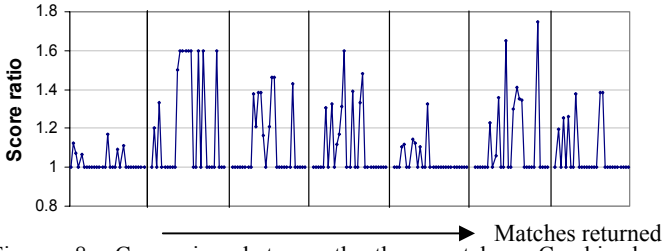

Figure 8. Comparison between the three matchers: Combined matcher improves precision.

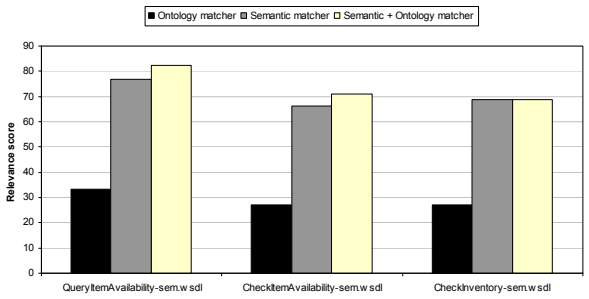

Figure 6. Comparison between the three matchers: Combined matcher improves precision.

References

[1] R. Akkiraju, R. Goodwin, P. Doshi, and S. Roeder. "A Method for Semantically Enhancing the Service Discovery Capabilities of UDDI". In the workshop proceedings of *Eighteenth International Joint Conference on Artificial Intelligence* 2003. Information Integration on the Web. WEB-1 pg: 87-92

[2] A. Ankolekar, M. Burstein, J. J. Hobbs, *et al*. "DAML-S: Semantic Markup for Web Services". In *Proceedings of the International Semantic Web Working Symposium (SWWS)* 2001.

[3] E. Christenson, F. Curbera, G. Meredith, and S. Weerawarana. "Web Services Description Language" (WSDL) 2001. www.w3.org/TR/wsdl

[4] IBM. "IBM Websphere UDDI Registry". 2003 http://www-3.ibm.com/software/webservers/appserv/was/network/

[5] OWL Technical Committee. "Web Ontology Language (OWL)". 2002. http://www.w3.org/TR/2002/WD-owl-ref-20021112/

[6] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. "Semantic Matching of Web Services Capabilities". *The First International Semantic Web Conference (ISWC)*, Sardinia (Italy), June, 2002.

[7] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. "Importing the Semantic Web in UDDI". *In Web Services, E-Business and Semantic Web Workshop*. 2002.

[8] UDDI Technical Committee. "Universal Description, Discovery and Integration (UDDI)". http://www.oasis-open.org/committees/uddi-spec/

[9] X. Dong et al. "Similarity search for web services," in Proc. VLDB, pp.372-283, Toronto, CA, 2004.

[10] A.Hess and N. Kushmerick, "Learning to attach metadata to web services," in Proc. Intl. Semantic web conference, 2003.

[11] J. Madhavan et al, "Generic schema matching with cupid," in Proc. VLDB 2001.

[12] S. Melnik et al, "Similarity flooding: A versatile graph matching algorithm and its application to schema matching," in Proc. ICDE, 2002.

[13] K. Sycara et al. "Dynamic service match making among agents in open information environments," in Jl. ACM SIGMOD Record, 1999.

[14] A.Patil et al. "Meteor-s web service annotation framework", in Proc. WWW conference, pp. 553-562, 2004.

[15] L. Li and I. Horrocks, " A software framework for matchmaking based on semantic web terminology," in Proc. WWW Conference, 2003.

[16] G.A. Miller, "WordNet: A lexical database for the English language," in Comm. ACM 1983.